Anti-Entropy Bandits for Geo-Replicated Consistency

Benjamin Bengfort and Pete Keleher
Department of Computer Science
University of Maryland, College Park, MD, USA
{bengfort,keleher}@cs.umd.edu

Abstract—**Eventually consistent systems can be made more consistent by reducing the time until a write is fully replicated, improving global visibility of updates. While gossip-based anti-entropy methods scale well, random selection of anti-entropy partners is less than efficient. Moreover, while eventual consistency may be consistent enough in a single data center, geographic replication increases visibility latency and leads to externally observable inconsistencies. In this paper, we explore an improvement to pairwise, bilateral anti-entropy; instead of uniform random selection, we introduce reinforcement learning mechanisms to assign selection probabilities to replicas most likely to have information. The result is more efficient replication, faster visibility, and stronger eventual consistency while maintaining high availability and partition tolerance.**

Introduction

A distributed system is made highly available when individual servers are allowed to operate independently without failure-prone, high latency coordination. The independent nature of the server's behavior means that it can immediately respond to client requests, but that it does so from a limited, local perspective which may be inconsistent with another server's response. If individual servers in a system were allowed to remain wholly independent, individual requests from clients to different servers would create a lack of order or predictability, a gradual decline into inconsistency, e.g. the system would experience *entropy*. To combat the effect of entropy while still remaining highly available, servers engage in periodic background *anti-entropy sessions* [17].

Anti-entropy sessions synchronize the state between servers ensuring that, at least briefly, the local state is consistent with a portion of the global state of the system. If all servers engage in anti-entropy sessions, the system is able to make some reasonable guarantees about consistent replication; the most famous of which is that without requests the system will become globally consistent, eventually [18]. More specifically, inconsistencies in the form of stale reads can be bound by likelihoods that are informed by the latency of anti-entropy sessions and the size of the system [2], [3]. Said another way, overall consistency is improved in an eventually consistent system by decreasing the likelihood of a stale read, which is tuned by improving the *visibility latency* of a write, the speed at which a write is propagated to a significant portion of servers. This idea has led many system designers to decide that eventual consistency is "consistent enough" [6], [19], particularly in a data center context where visibility latency is far below the rate of client requests, leading to practically strong consistency.

However, there has been an important change in considerations for the design of such systems that have led us to re-evaluate propagation speed: systems are growing, while simultaneously becoming geographically distributed outside of datacenters. Large geographically-distributed systems are becoming the norm particularly as mobile devices and sensor systems participate in computing and storage at the edge of large distributed systems [9], [16]. From content delivery systems that span the globe, to mobile applications, to future systems such as automated vehicular networks, all will require additional consistency guarantees without sacrificing availability. However, scaling an eventually consistent system to dozens or even hundreds of nodes increases the radius of the network, which leads to increased noise during anti-entropy e.g. the possibility that an anti-entropy session will be between two already synchronized nodes. Geographic distribution and extra-datacenter networks also increase the latency of anti-entropy sessions so that inconsistencies become more apparent to external observers.

We address the challenge of large, geographically distributed eventually consistent systems by improving synchronization using reinforcement learning techniques. Anti-entropy uses gossip and rumor spreading to propagate updates deterministically without saturating the network even in the face of network outages [10], [11], [14]. These protocols use uniform random selection to choose synchronization peers, which means that a write occurring at one replica is not efficiently propagated across the network. In this paper we explore the use of *multi-armed bandit* algorithms [12], [13] to optimize for fast, successful synchronizations by modifying peer selection probabilities. The result is a synchronization topology that emerges according to access patterns and network latency which produces efficient synchronization, localizes most data exchange, lowers visibility latency and increases consistency.

Our contribution for this early stage work is a demonstration of the potential for replicas to meaningfully influence global consistency by modifying local behavior in response to their computing environment. This potential has been motivated by our larger work which investigates the effect of scaling systems, both in terms of size and distance, on consistency. We show this potential though experiments run on a system with dozens of replica distributed across 5 continents which show that even a relatively simple implementation of adaptivity leads to a pronounced benefit in visibility latency, and therefore the overall consistency of the system.

Our investigation considers an eventually consistent, in-memory key-value store that is totally replicated using anti-entropy [8]. A brief description of the system and consistency considerations follows.

Accesses and Consistency

Clients can `Put` (write) and `Get` (read) key-value pairs to and from one or more replicas in a single operation. The set of replicas that responds to a client creates a quorum that must agree on the state of the operation at its conclusion. Clients can vary read and write quorum sizes to improve consistency or availability – larger quorums reduce the likelihood of inconsistencies caused by concurrent updates, but smaller quorums respond much more quickly, particularly if the replicas in the quorum are co-located with the client. In large, geo-replicated systems we assume that clients will prefer to choose fewer, local replicas to connect with, optimistic that collisions across the wide-area are rare, e.g. that writes are localized but reads are global.

On `Put`, the instance of the key-value pair created by the update is assigned a monotonically increasing, conflict-free *version number* [15], [1]. For simplicity, we assume a fixed number of replicas, therefore each version is made up of two components: the *update* and *precedence ids*. Precedence ids are assigned to replicas during configuration, and update ids are incremented to the largest observed value during synchronization. As a result, any two versions generated by a `Put` anywhere in the system are comparable such that the *latest* version of the key-value pair is the version with the largest update id, and in the case of ties, the largest precedence id.

Additional version metadata, including the parent version of the update (in a read-then-write system or simply the latest version of the key stored locally), implements a virtual object history that allows us to reason about consistency. Keys can be managed independently, e.g. each key has its own update id sequence resulting in per-object consistency, or all objects can be managed together with a single sequence; in the latter case, it is possible to construct an ordering history of operations to all objects and in the former, a sequence of operations for each object. Object histories allow us to reason about the global consistency of the system.

There are two primary inconsistencies that can occur in this system: *stale reads* and *forked writes*. A stale read means that the `Get` operation has not returned the globally most recent version of the object, e.g. the local replica is behind in the object history. A forked write is caused when there are two concurrent writes to the same object, a symptom of stale reads. Forked writes cause a divergence in the object history such that there are two or more branches of update operations. As we will see in the next section, one of these writes will eventually be *stomped* before it can become fully replicated, meaning that the eventual consistency prunes these branches at the cost of losing the update. The ideal consistency for a system is represented by a linear object history without forks [7], which demonstrates that the system was in a consistent state during all accesses.

Both forms of inconsistency can be primarily attributed to *visibility latency*, that is the time it takes for an update to propagate to all replicas in the system. Visibility latency is directly related to the likelihood of stale reads with respect to the frequency of accesses [3]; said another way, decreasing the visibility latency improves the overall consistency of a system. However, in a system that uses anti-entropy for replication, the propagation speed of an update is not governed solely by network connections, it is also bound to the number and frequency of anti-entropy sessions conducted and the radius of the network.

Anti-Entropy

Anti-entropy sessions are conducted in a pairwise fashion on a periodic interval to ensure that the network is not saturated with synchronization requests which may reduce client availability. At each interval, every replica selects a synchronization partner such that all replicas have a uniform likelihood of selection. This ensures that an update originating at one replica will be propagated to all online replicas given the continued operation of replication. This mechanism also provides robustness in the face of failure; a single unresponsive replica or even network partition does not become a bottleneck to synchronization, and once the failure is repaired synchronization will occur without reconfiguration.

There are two basic forms of synchronization: *push* synchronization is a fire-and-forget form of synchronization where the remote is sent the latest version of all objects, whereas *pull* synchronization requests the latest version of objects and minimizes the size of data transfer. To get the benefit of both, we consider *bilateral* synchronization which combines push and pull in a two-phase exchange. Bilateral synchronization increases the effect of anti-entropy during each exchange because it ensures that in the common case each replica is synchronized with two other replicas instead of one during every anti-entropy period.

Bilateral anti-entropy starts with the initiating replica sending a vector of the latest local versions of all keys currently stored, usually optimized with Merkel or prefix trees to make comparisons faster. The remote replica compares the versions sent by the initiating replica with its current state and responds with any objects whose version is *later* than the initiating replica's as well as another version vector of requested objects that are earlier on the remote. The initiating replica then replies with the remote's requested objects, completing the synchronization. We refer to the first stage of requesting later objects from the remote as the pull phase, and the second stage of responding to the remote the push phase.

There are two important things to note about this form of anti-entropy exchange. First, this type of synchronization implements a *latest writer wins* policy. This means that not all versions are guaranteed to become fully replicated – if a later version is written during propagation of an earlier version, then the earlier version gets *stomped* by the later version

because only the latest versions of objects are exchanged. If there are two concurrent writes, only one write will become fully replicated, the write on the replica with the greater precedence. Second, visibility latency is maximized when all replicas choose a remote synchronization partner that does not yet have the update. This means that maximal visibility latency is equal to $t \log_3 n$, where t is the anti-entropy interval and n is the number of replicas in the network. In practice, however, because of inefficient exchanges due to uniform random selection of synchronization partners, this latency is never practically achieved, and is instead modulated by a noise variable that is proportional to the size of the network.

BANDIT APPROACHES

To combat the effect of noise on visibility latency our initial approach employs a technique commonly used in active and reinforcement learning: multi-armed bandits. Multi-armed bandits refer to a statistical optimization procedure that is designed to find the optimal payout of several choices that each have different probabilities of reward. In this case, we use bandits to improve uniform random selection of peers so that replicas choose synchronization partners that are most likely to exchange information and thus more quickly propagate updates, while still maintaining the properties of full replication and fault tolerance.

A bandit problem is designed by identifying several (usually more than two) competing choices called "arms"[1], as well as a reward function that determines how successful the selection of an arm is. During operation, the bandit selects an arm, observes the rewards, then updates the payout likelihood of the selected arm, normalized by the number of selections. As the bandit selects arms, it learns which arm or arms have the highest likelihood of reward, and can modify it's arm selection *strategy* to maximize the total reward over time.

Bandits must balance exploration of new arms with possibly better reward values and exploitation of an arm that has higher rewards than the other. In the *epsilon greedy strategy*, the bandit will select the arm with the best reward with some probability $1 - \epsilon$, otherwise it will select any of the arms with uniform probability. The smaller ϵ is, the more the bandit favors exploitation of known good arms, the larger ϵ is, the more it favors exploration. If $\epsilon = 1$ then the algorithm is simply uniform random selection. A simple extension of this is a strategy called *annealing epsilon greedy*, which starts with a large ϵ, then as the number of trials increases, steadily decreases ϵ on a logarithmic scale. There are many other bandit strategies but we have chosen these two simple strategies for our initial research to demonstrate a bolt-on effective improvement to existing systems.

Peer selection for anti-entropy is usually conducted with uniform random selection to guarantee complete replication. To extend anti-entropy with bandits, we design a selection method whose arms are remote peers and whose rewards are

---

[1]Arms refer to the pulling mechanism of a slot machine, the metaphor generally used to motivate the multi-armed bandit problem.

	Pull	Push	Total
Synchronize at least 1 object	0.25	0.25	0.50
Synchronize multiple objects	0.05	0.05	0.10
Latency $\leq$ 5ms (local)	0.10	0.10	0.20
Latency $\leq$ 100ms (regional)	0.10	0.10	0.20
Total	*0.50*	*0.50*	*1.00*

TABLE I: Reward Function

determined by the success of synchronization. The goal of adding bandits to anti-entropy is to optimize selection of peers such that the visibility latency becomes closer to the optimal propagation time as a synchronization topology emerges from the bandits. A secondary goal is to minimize anti-entropy latency by preferring local (in the same data center) and regional (e.g. on the same continent) connections.

Our initial reward function favors synchronizations to replicas where the most writes are occurring by giving higher rewards to anti-entropy sessions that exchange later versions in either a push or a pull as well as additional rewards if more than one object is exchanged. Additionally, the latency of the synchronization RPCs is computed to reward replicas that are near each other. The complete reward function is given in Table I: for each phase of synchronization (push and pull), compute the reward as the sum of the propositions given. For example if a synchronization results in three objects being pulled in 250ms, and one object being pushed in 250ms, the reward is 0.75.

The design of reward functions can be implemented to the needs of a specific system. For example, in a system that has workloads with variable sized writes, object size could be considered or systems with imbalanced deployments might consider a reward function that prioritizes inter-region communication.
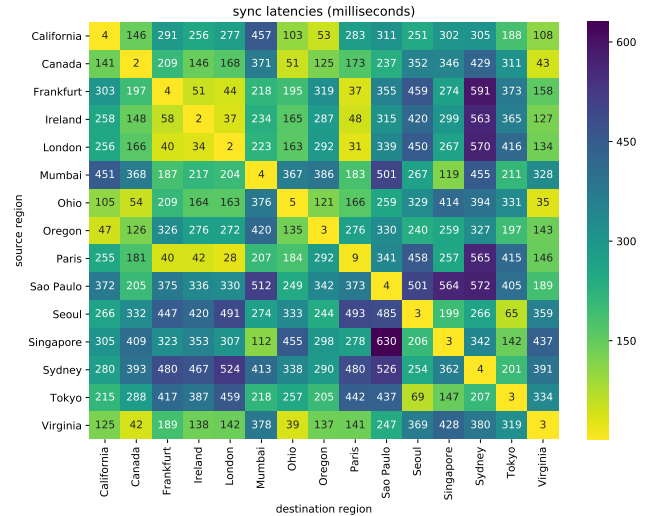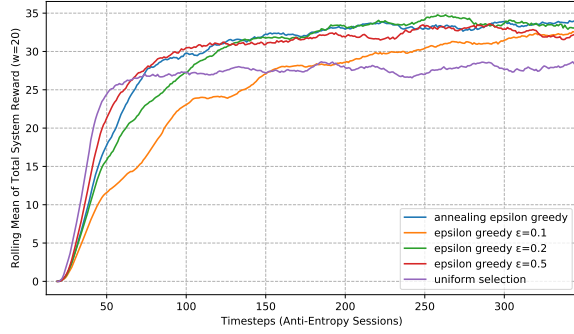
sync latencies (milliseconds)

source region \ destination region	California	Canada	Frankfurt	Ireland	London	Mumbai	Ohio	Oregon	Paris	Sao Paulo	Seoul	Singapore	Sydney	Tokyo	Virginia
California	4	146	291	256	277	457	103	53	283	311	251	302	305	188	108
Canada	141	2	209	146	168	371	51	125	173	237	352	346	429	311	43
Frankfurt	303	197	4	51	44	218	195	319	37	355	459	274	591	373	158
Ireland	258	148	58	2	37	234	165	287	48	315	420	299	563	365	127
London	256	166	40	34	2	223	163	292	31	339	450	267	570	416	134
Mumbai	451	368	187	217	204	4	367	386	183	501	267	119	455	211	328
Ohio	105	54	209	164	163	376	5	121	166	259	329	414	394	331	35
Oregon	47	126	326	276	272	420	135	3	276	330	240	259	327	197	143
Paris	255	181	40	42	28	207	184	292	9	341	458	257	565	415	146
Sao Paulo	372	205	375	336	330	512	249	342	373	4	501	564	572	405	189
Seoul	266	332	447	420	491	274	333	244	493	485	3	199	266	65	359
Singapore	305	409	323	353	307	112	455	298	278	630	206	3	342	142	437
Sydney	280	393	480	467	524	413	338	290	480	526	254	362	4	201	391
Tokyo	215	288	417	387	459	218	257	205	442	437	69	147	207	3	334
Virginia	125	42	189	138	142	378	39	137	141	247	369	428	380	319	3

Fig. 1: Inter-Region Synchronization Latencies (Push+Pull)

Fig. 2: Total system rewards over time

Fig. 3: Visibility Latency by Region

EXPERIMENTS

We conducted experiments using a distributed key-value store totally replicated across 45 replicas in 15 geographic regions on 5 continents around the world. Replicas were hosted using AWS EC2 t2.micro instances and were connected to each other via internal VPCs when in the same region, using external connections between regions. The store, called Honu, is implemented in Go 1.9 using gRPC and protocol buffers for RPC requests; all code is open source and available on GitHub.

The workload on the system was generated by 15 clients, one in each region and colocated with one of the replicas. Clients continuously created Put requests for random keys with a unique prefix per-region such that consistency conflicts only occur within a single region. The average throughput generated per-client was 5620.4 puts/second. The mean synchronization latency between each region ranged from 35ms to 630ms, shown in 1, because of this we set the anti-entropy interval to 1 second to train the system, then set the interval to 125ms to measure visibility latency. To account for lag between commands sent to replicas in different regions, each experiment was run for 11 minutes, the bandit learning period was 4 minutes then visibility latency was observed for 6 minutes, buffered by 30 seconds before and after the workload to allow replicas to initialize and gracefully shutdown.

Our first experiments compared uniform random peer selection with epsilon greedy bandits using $\epsilon \in \{0.1, 0.2, 0.5\}$ as well as an annealing epsilon greedy bandit. The total system rewards as a rolling mean over a time window of 20 synchronizations are shown in Figure 2. The rewards ramp up from zero as the clients come online and start creating work to be synchronized. All of the bandit algorithms eventually improve over the baseline of uniform selection, not only generating more total reward across the system, but also introducing less variability in rewards over time. None of the bandit curves immediately produces high rewards as they explore the reward space; lower ϵ values may cause exploitation of incorrect arms, while higher ϵ values take longer to find optimal topologies. However, in the static workload case, the more aggressive bandit strategies converge more quickly to the optimal reward.
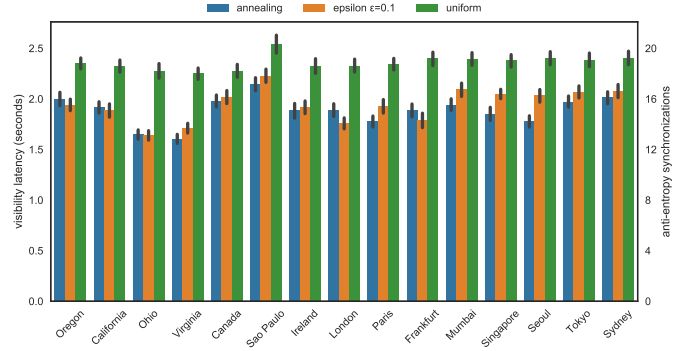
Visibility latencies were computed by reducing the workload rate to once every 4 seconds to ensure the write becomes fully visible across the entire network. During the visibility measurement period, replicas logged the timestamp the write was pushed or pulled, and the visibility latency is computed as the difference between the minimum and maximum timestamp. Both the visibility latency and the average number of anti-entropy sessions are shown in Figure 3. Employing bandit strategies reduces the visibility latency from 2360ms on average in the uniform case to 1870ms, reducing the number of required anti-entropy intervals by approximately 4.

To show the emergent behavior of bandits, we have visualized the resulting topologies as network diagrams in Figure 4 (uniform selection), Figure 5 (annealing epsilon) and Figure 6 (epsilon greedy $\epsilon = 0.1$). Each network diagram shows each replica as a vertex, colored by region e.g. purple is California, teal is Sao Paulo, Brazil, etc. Each vertex is also labeled with the 2-character UN country or US state abbreviation as well as the replica's precedence id. The size of the vertex represents the number of Put requests that replica received over the course of the experiment; larger vertices represent replicas that were colocated with workload generators. Each edge between vertices represents the total number of successful synchronizations, the darker and thicker the edge is, the more synchronizations occurred between the two replicas. Edges are directed, the source of the edge is the replica that initiated anti-entropy with the target of the edge.

Comparing the resulting networks, it is easy to see that more defined topologies result from the bandit-based approaches. The uniform selection network is simply a hairball of connections with a limited number of synchronizations. Clear optimal connections have emerged with the bandit strategies, dark lines represent extremely successful synchronization connections between replicas, while light lines represent synchronization pairs that are selected less frequently. We posit that fewer edges in the graph represents a more stable network; the fewer synchronization pairs that are selected, the less noise that occurs from selecting a peer that is in a similar state.
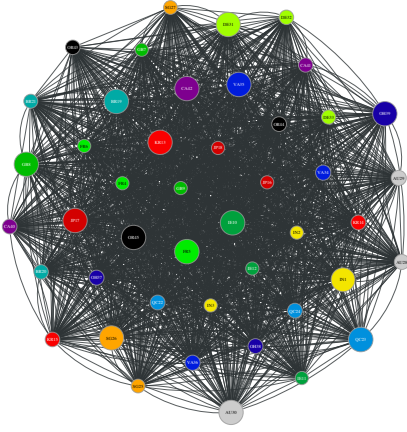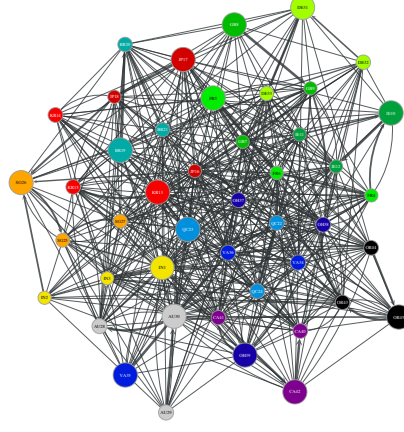
Fig. 4: Uniform Selection
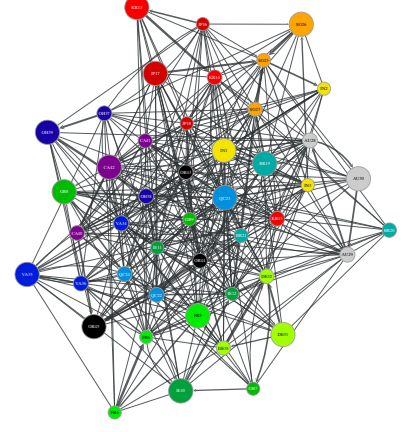

Fig. 5: Annealing Epsilon Greedy

Fig. 6: Epsilon Greedy $\epsilon = 0.1$

DISCUSSION

To achieve stronger eventual consistency, the visibility latency of a system replicated with anti-entropy must be reduced. We believe that this can be achieved with two primary goals: increasing the number of successful synchronizations and maximizing the number of local and regional synchronizations such that the average latency of anti-entropy sessions is as low as possible. These goals must also be tempered against other requirements, such as fault and partition tolerance, a deterministic anti-entropy solution that ensures the system will become consistent eventually, and load balancing the synchronization workload evenly across all replicas.

Bandit based approaches to peer selection clearly reduce noise inherent in uniform random selection as shown in Figure 2. The bandit strategies achieve better rewards over time because peers are selected that are more likely to have an update to synchronize. Moreover, based on the network diagrams shown in Figures 4-6, this is not the result of one or two replicas becoming primary syncs: most replicas have only one or two dark in-edges meaning that most replicas are only the most valuable peers for one or two other replicas.

Unfortunately, the rewards using a bandit approach, while clearly better than the uniform case, are not significantly better – this is an interesting demonstration of the possibility of adaptive systems to improve consistency but further investigation is required. The primary place we see for adjustment is future work to explore the reward function in detail. For example, the inclusion of penalties (negative rewards) might make the system faster to adjust to a high quality topology. Comparing reward functions against variable workloads may also reveal a continuum that can be tuned to the specific needs of the system.

As for localization, there does appear to be a natural inclination for replicas that are geographically proximate to be a more likely selection. In Figure 6, replicas in Canada (light blue), Virginia (dark blue), Sydney (grey), California (purple), and Frankfurt (light green) all prioritize local connections. Regionally, this same figure shows strong links such as those

between Ohio and California (`CA42` → `OH38`) or Japan and Singapore (`JP17` → `SG25`). Replicas such as `BR19` and `IN3` appear to be hubs that specialize in cross-region collaboration. Unfortunately there does also seem to be an isolating effect, for example Sydney (grey) appears to have no significant out of region synchronization partners. Multi-stage bandits might be used to create a tiered reward system to specifically adjust the selection of local, regional, and global peers. Other strategies such as upper confidence bounds, softmax, or Bayesian selection may also create more robust localization.

Finally, and perhaps most significantly, the experiments conducted in this paper were on a static workload; future work must explore dynamic workloads with changing access patterns to more closely simulate real world scenarios. While bandit algorithms are considered online algorithms that do respond to changing conditions, the epsilon greedy strategy can be slow to change since it prefers to exploit high-value arms. Contextual bandits use side information in addition to rewards to make selection decisions, and there is current research in exploring contextual bandits in dynamic worlds that may be applicable [13]. Other strategies such as periodic reseting of the values may incur a small cost to explore the best anti-entropy topology, but could respond to changing access patterns or conditions in a meaningful way.

CONCLUSION

In this paper we have presented a demonstration of adaptive consistency in the geo-replicated eventually consistent systems by employing a novel approach to peer selection during anti-entropy – replacing uniform random selection with multi-armed bandits. Multi-armed bandits consider the historical reward obtained from synchronization with a peer, defined by the number of objects synchronized and the latency of RPCs, when making a selection. Bandits balance the exploitation of a known high-value synchronization peer with the exploration of possibly better peers or the impact of failures or partitions. The end result is a replication network that is less perturbed

by noise due to randomness and capable of more efficiently propagating updates.

In an eventually consistent system, efficient propagation of updates is directly tied to higher consistency. By reducing visibility latency, the likelihood of a stale read decreases, which is the primary source of inconsistency in a highly available system. We have demonstrated that bandit approaches do in fact lower visibility latency in a large network.

This work, however, is preliminary. Future efforts will consider different reward functions, different selection strategies, dynamic environments, and how the priorities of system designers can be embedded into rewards. Reward functions that capture more information about the expected workload of the system such as object size, number of conflicts, or localizing objects may allow specific tuning of the adaptive approach. We will also specifically explore in detail the effect of dynamic workloads on the system and how the reinforcement learning can adapt in real time to changing conditions. We plan to investigate periodic resets, anomaly detection, and auction mechanisms to produce efficient topologies that are not brittle as access patterns change. We also plan to evaluate other reinforcement learning strategies such as neural or Bayesian networks to determine if they handle dynamic environments more effectively.

We believe that the results presented show a promising start to a renewed investigation of highly available distributed storage systems in novel network environments, particularly those that span the globe. Specifically, this work is part of a larger exploration of adaptive, globally distributed data systems that federate consistency levels to provide stronger guarantees [5]. Federated consistency combines adaptive eventually consistent systems such as the one presented in this paper with scaling geo-replicated consensus such as Hierarchical Consensus [4] in order to create robust data systems that are automatically tuned to provide the best availability and consistency. Distributed systems that adapt to and learn from their environments and access patterns, such as the emerging synchronization topologies we observed in this paper, may form the foundation for the extremely large, extremely efficient networks of the future.

All code for the key-value store and bandit-based entropy as well as experimental results is open source and available on GitHub at https://github.com/bbengfort/honu.

References

[1] Paulo Srgio Almeida, Carlos Baquero, and Victor Fonte. Version stamps-decentralized version vectors. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 544–551. IEEE.

[2] Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Probabilistically bounded staleness for practical partial quorums. 5(8):776–787.

[3] Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Quantifying eventual consistency with PBS. 23(2):279–302.

[4] Benjamin Bengfort and Pete Keleher. Brief announcement: Hierarchical consensus. In *Proceedings of the 2017 ACM Symposium on Principles of Distributed Computing*, pages 355–357. ACM. bibtex: bengfort2017brief.

[5] Benjamin Bengfort and Pete Keleher. Federating consistency for partition-prone networks. In *Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE.

[6] David Bermbach and Stefan Tai. Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior. In *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing*, page 1. ACM.

[7] Philip A. Bernstein and Sudipto Das. Rethinking eventual consistency. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 923–928. ACM.

[8] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM.

[9] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing: Vision and challenges. 45(5):37–42.

[10] Bernhard Haeupler. Simple, fast and deterministic gossip and rumor spreading. 62(6):47.

[11] Richard Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vocking. Randomized rumor spreading. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 565–574. IEEE.

[12] John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in Neural Information Processing Systems*, pages 817–824. bibtex: langford_epoch-greedy_2008.

[13] Haipeng Luo, Alekh Agarwal, and John Langford. Efficient contextual bandits in non-stationary worlds.

[14] Yamir Moreno, Maziar Nekovee, and Amalio F. Pacheco. Dynamics of rumor spreading in complex networks. 69(6):066130.

[15] D. Stott Parker, Gerald J. Popek, Gerard Rudisin, Allen Stoughton, Bruce J. Walker, Evelyn Walton, Johanna M. Chow, David Edwards, Stephen Kiser, and Charles Kline. Detection of mutual inconsistency in distributed systems. (3):240–247.

[16] Salvatore Scellato, Cecilia Mascolo, Mirco Musolesi, and Jon Crowcroft. Track globally, deliver locally: improving content delivery networks by tracking geographic social cascades. In *Proceedings of the 20th international conference on World wide web*, pages 457–466. ACM.

[17] Douglas B. Terry, Alan J. Demers, Karin Petersen, Mike J. Spreitzer, Marvin M. Theimer, and Brent B. Welch. Session guarantees for weakly consistent replicated data. In *Parallel and Distributed Information Systems, 1994., Proceedings of the Third International Conference on*, pages 140–149. IEEE.

[18] Werner Vogels. Eventually consistent. 52(1):40–44.

[19] Hiroshi Wada, Alan Fekete, Liang Zhao, Kevin Lee, and Anna Liu. Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective. In *CIDR*, volume 11, pages 134–143.